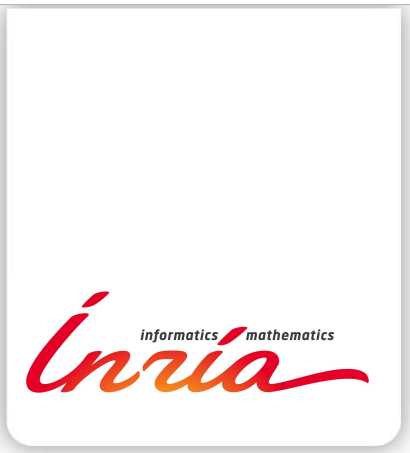
MineSweeper: Where to Probe?

Marc Legendre, Kévin Hollard, Olivier Buffet, Alain Dutech

RESEARCH REPORT
N° 8041
July 2012
Project-Team MAIA

ISSN 0249-6399 ISRN INRIA/RR--8041--FR+ENG



MineSweeper: Where to Probe?

Marc Legendre, Kévin Hollard, Olivier Buffet, Alain Dutech

Project-Team MAIA

Research Report n° 8041 — July 2012 — 26 pages

Abstract: Most research about the game of Minesweeper has focussed on inferring which cells may or may not contain a mine, discussing the complexity of this problem (and its variants) and proposing efficient resolution techniques. While this inference task is indeed crucial to playing Minesweeper, this paper comes back to the original game as a whole, modelling it as a sequential decision-making problem —more precisely as a Partially Observable Markov Decision Process (POMDP)— before proposing and studying various heuristics to solve the key problem of selecting the next cell to play.

Key-words: Minesweeper, POMDP, heuristics

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Démineur: Où sonder?

Résumé : La majeure partie de la recherche sur le jeu du démineur s'est concentrée sur le problème d'inférer quelles cellules peuvent ou ne peuvent pas contenir une mine, discutant de la complexité de ce problème (et de ses variantes) et proposant des techniques de résolution efficaces. Si cette tâche d'inférence est en effet cruciale pour jouer au démineur, ce document revient au jeu original dans sa totalité, le modélisant comme un problème de prise de décision séquentielle – plus précisément comme un processus de décision markovien partiellement observable (POMDP)– avant de proposer et d'étudier diverses heuristiques pour résoudre le problème clef de la sélection de la prochaine cellule à jouer.

Mots-clés : démineur, PDMPO, heuristiques

Contents

1 Introduction

In the game of Minesweeper, a single player faces a grid representing a random minefield. Whenever he picks ("clicks in") a cell c, the game terminates if this cell contains a mine, or an area is uncovered, starting from c, stopping at cells having at least one adjacent mine, and with the number of adjacent mines marked on each uncovered cell. Figure 1 shows the result of selecting the cell c in the bottom-left corner of an 8×8 minefield. The process is repeated until the game ends (i) successfully if only mined cells remain, or (ii) unsuccessfully if a mined cell is picked.

An important computational effort is typically put into figuring out which cells may or may not contain a mine given the observed values (or even computing the probability to contain a mine). One usually reasons about the cells for which observed values in the vicinity give some information, i.e., cells marked with a '?' in Fig. 1. Such a task amounts to solving a Constraint Satisfaction Problem (CSP) and has drawn most of the attention, with a lot of research discussing the complexity of solving such a CSP (or one of its many variants), and on proposing efficient —possibly approximate— resolution techniques.

While this inference task is indeed crucial to playing Minesweeper, we come back to the original game as a whole, modelling it as a sequential decision-making problem —more precisely as a Partially Observable Markov Decision Process (POMDP)— and proposing an efficient heuristic to solve the key problem of selecting the next cell to play.

This paper starts in its Background section by detailing the rules of the Minesweeper game (Section 2.1), describing related work (Section 2.2), and explaining how this game translates into a POMDP (Section 2.3). Because this problem cannot be solved with state-of-the-art algorithms, we then look more closely at the game at hand to derive a specific approach, presented in Section 3, which we evaluate through experiments in Section 4. A discussion concludes this paper (Section 5).

2 Background

2.1 Game Rules

This section first details the —classical— game rules we consider in this work, before mentioning some possible variants.

2.1.1 Chosen Rules

An initial game configuration for Minesweeper is defined by a tuple $\langle W, H, M \rangle$ where:

- $W > 0$ and $H > 0$ are the integer *width* and *height* of the minefield (a grid of size $W \times H$), and
- a $W \times H$ matrix M of zeros and ones —called *minefield*— is generated so as to contain m ones (=mines), meaning that the number of possible minefields is:

$$|\mathcal{M}_{W \times H, m}| = \left(\begin{array}{c} W \cdot H \\ m \end{array} \right).$$

Let us also define:

- the *adjacent cells* $\alpha(c)$ of a cell $c = (w, h)$ as the set of (up to 8) cells $c' = (w', h')$ $(\neq c)$ such that $|w - w'| \leq 1$ and $|h - h'| \leq 1$;

	1	2	3	4	5	6	7	8
1	.	.	.	.	.	.	.	.
2	?	?	?	.	.	.	.	.
3	1	2	?	.	.	.	.	.
4		1	?	?	?	?	.	.
5		1	1	1	2	?	.	.
6					2	?	.	.
7				1	2	?	.	.
8	c			1	?	?	.	.

Figure 1: Example showing a cell's c neighborhood $N(c)$, its interior (empty cells + 'c'), boundary (numbered cells), fringe ('?'), and un-sensed cells ('.') in an 8×8 minefield.

- the *neighborhood* $N(c)$ of a cell $c = (w, h)$ as the set containing
  - the cell c if $M_c = 0$, plus,
  - if $\sum_{c' \in \alpha(c)} M_{c'} = 0$, c's adjacent cells' neighborhoods;

  this recursive definition corresponds to an expansion through the minefield as long as cells with no adjacent mines are encountered; note that, if c contains a mine, then $N(c) = \emptyset$;
- the *interior* $N^o(c)$ of a neighborhood $N(c)$ as the set of cells c' in $N(c)$ which are not adjacent to a mine (e.g., such that $\sum_{c' \in \alpha(c)} M_{c'} = 0$);
- the *boundary* (or *frontier*) $\partial N(c)$ of a neighborhood $N(c)$ as the set of cells c' in $N(c)$ which are adjacent to at least one mine (e.g., such that $\sum_{c' \in \alpha(c)} M_{c'} > 0$);
- the *fringe* $\phi(c)$ of a neighborhood $N(c)$ as the set of cells which are adjacent to some cell in $N(c)$ but are not in $N(c)$;
- the *un-sensed* cells $U(c)$ as the set of cells not in one of the aforementioned sets.

Given a finite set of cells $C = \{c_1, \cdots, c_{|C|}\}$, these notations naturally extend to define the neighborhood $N(C)$ (union of individual neighborhoods), the interior $N^o(C)$ (union of individual interiors), the boundary $\partial N(C)$ (union of individual boundaries), the fringe $\phi(C)$ (neither union nor intersection of individual fringes), or the un-sensed cells $U(C)$ (intersection of individual sets of un-sensed cells). Note that, for convenience, we abusively employ terminology from point-set topology. Usual definitions or properties linked to these notions may not hold.

The player —who does not know the content of M— is initially provided with a $W \times H$ *knowledge* matrix K (initially filled with -1s indicating no certain knowledge). The game then consists in repeating the following steps:

- the player picks as its action an uncovered cell $a_t \in \{1..W\} \times \{1..H\}$, i.e., a cell of (observed) value $K_{a_t} = -1$;
- if $M_{a_t} = 1$, then end the game with failure;
- otherwise, update the value of K_{a_t} with the number of mines surrounding this cell;[1]

[1] Note that in "human minesweeper", K is updated for all of c's neighborhood. In practice, a very simple rule easily compensates for this difference.

- if only m cells with value -1 remain in K, then end the game successfully.

Note that the knowledge matrix only reflects observations, not deductions. One may really "know" (by deduction) that some cell c is mined even though $K_c = -1$.

The objective is to select a sequence of actions maximizing the probability to end the game successfully. Possible secondary objectives —which we will consider only occasionally— are (i) to minimize the number of actions, and (ii) to minimize the computation time (including pre-computations) and/or memory usage.

Finally, because they will be used in the remainder of this paper, we also introduce the following notations:

- $p_m(c)$ is the probability for a cell c to host a mine: $p_m(c) = Pr(\text{mine in cell } c|K)$;
- a cell is *safe* iff $p_m(c) = 0$;
- a cell is *mined* iff $p_m(c) = 1$.

2.1.2 Possible Variants

Note that possible variants exist. We here mention three important points that can distinguish variants:

- **First move:** Official versions of Minesweeper —for human competitors— ensure that the minefield M is generated after the first move, while making sure that the chosen cell is empty. This prevents frustrating 1-click loses.

  This assumption will be made in our experiments, which will allow to compare results with previous automated solvers.

- **Topology:** The classical Minesweeper game takes place on a 2D rectangular grid, but other graph topologies can be considered. The present work can be straightforwardly extended to other graph topologies as long as the number of cells is finite.

- **Distribution of mines:** Instead of specifying a fixed number of mines m, one could specify the probability $\rho \in (0, 1)$ for any cell to contain a mine. To end the game, the player should then indicate when he believes that all empty cells have been uncovered (i.e., only mined cells remain), what may not be known with certainty.

2.2 Related Work

In this section, we both discuss basic solution techniques and related work. Note that a good source of bibliographical references is the website [MineSweeper].

Human or artificial players typically put a lot of effort into deducing information —where mines can be— from observations —the knowledge matrix K—. After this *inference* step, one can select the next cell to play depending on the acquired information. This *decision* step usually relies on simple decision rules.

2.2.1 Building Blocks and Strategies

In the remainder of this paper, we will describe a number of "small" algorithms —typically denoted with 2–3 letter abbreviations— that serve as building blocks for complete playing algorithms (referred to as "strategies") —typically denoted as the concatenation of 2–3 letter abbreviations (in order of priority/execution).

Note that we will distinguish three types of building blocks:

- *first-move* building blocks that make decisions when facing a new minefield;
- *non-decisive* building blocks that may not pick a cell and let a (lower priority) building block make a choice; and
- *tie-breaking* building blocks that necessarily pick a cell (and should therefore have the lowest priority).

A strategy of course contains exactly one first-move building block and one tie-breaking building block, while there may be any number of non-decisive building blocks.

2.2.2 Inference as a Constraint Satisfaction Problem

Two obvious decision rules are (i) do not pick a (necessarily) mined cell, and (ii) pick a safe cell if one exists.[2] Thus, a first natural inference task is to determine which cells are mined and which cells are safe.

Exact Inference For a given cell c, determining whether it is mined (respectively safe) can be achieved by assuming it is not, and searching for a configuration of mines compatible with the observation K, i.e., solving a specific Constraint Satisfaction Problem (CSP). If no solution is found, then the cell is mined (respectively safe). Yet, rather than following this process for each cell separately, one can —making no assumptions on mined or safe cells— enumerate all configurations compatible with K, and then deduce for each cell c whether it is safe (no configuration where c is mined) or mined (c is mined in all configurations). We are thus facing a *model counting* problem . In addition, one can compute the probability for c to be mined as the ratio

$$p_m(c) = \frac{\#(\text{cfgs where } c \text{ is mined})}{\#(\text{all cfgs})}.$$

Searching for a single configuration compatible with K can be formulated as a CSP as follows:

$$\forall c = (i,j),\ x_{i,j} \in \{0,1\} \qquad (x_{i,j} \text{ is either mined or safe}),$$

$$\forall c = (i,j),\ (K_{i,j} \geq 0) \Rightarrow (x_{i,j} = 0) \qquad (\text{a cell } c \text{ with non-negative value is safe}),$$

$$\forall c = (i,j),\ (K_{i,j} \geq 0) \Rightarrow \left(\sum_{c'=(i',j') \in \alpha(c)} x_{i',j'} = K_{i,j} \right)$$

(the non-negative value on a cell c is the number of adjacent mined cells),

$$\sum_{c=(i,j)} x_{i,j} = m \qquad (\text{there are exactly } m \text{ mined cells}).$$

This is a hard problem as it has been proven to be NP-complete [Kaye, 2000].[3] This justifies developing efficient resolution techniques, possibly exploiting the structure of the problem at hand. Here are some key ideas:

[2] Picking a safe cell may not be the best move if (i) optimizing the secondary criterion "number of moves" and (ii) picking this cell will provably not provide any new information.

[3] Thus, since the inference problem in Minesweeper is NP-complete, anyone finding a polynomial-time algorithm to solve it would win a million Dollar [Stewart, 2000].

- some simple rules —considered as one building block algorithm and denoted `Sim`— allow determining mined or safe cells quickly; for example, on Fig. 1: $(3,4)$ necessarily contains a mine (because it is the only fringe cell among adjacent cells of $(2,5)$, and $K_{2,5} = 1$); then, it follows that $(3,3)$ and $(4,4)$ are safe cells (because $(2,4)$ and $(3,5)$ already have the number of adjacent mines they require); these simple deductions (plus some others) are shown on Fig. 2;

- given the set C of previously played cells, $N(C)$ is the set of uncovered (=numbered) cells in K; the interior $N^o(C)$ does not provide any information on where mines may be lying, while the boundary $\partial N(C)$ does;

- the values observed on the boundary $\partial N(C)$ are related to mines in the fringe $\phi(C)$; so, we can restrict the reasoning about constraints induced by K to the fringe;

- the total number of mines #(mines) in the grid constrains the number of mines in the fringe:

$$\begin{aligned} \#(\text{mines in } \phi(C)) &\leq \#(\text{mines}), \\ &\geq \#(\text{mines}) - |U(C)| \end{aligned}$$
(because all remaining mines should fit in $\phi(C) \cup U(C)$);

- one can often split the problem —possibly thanks to acquired knowledge about safe or mined cells— into independent CSPs, what may dramatically reduce the problem's complexity; in Fig. 2, having inferred the status of 6 fringe cells, we are left with two problems: one about $\{(1,2),(2,2)\}$, the other about $\{(6,4),(6,5),(6,6),(6,7),(6,8)\}$;

- also, for a given fringe cell c, the information carried by cells close to it (mainly adjacent cells) may give a hint on c's content; one should then reason first on cells with the highest "certainty" to be either mined or safe as they should help quickly pruning the search;

- finally, given a possible configuration of mines in the fringe C', the probability to have a mine outside of the fringe is, for any cell c:

$$p_m(c) = \frac{\#(\text{mines}) - \#(\text{mines in } C')}{|U(C)|}.$$

More details on how to efficiently solve this CSP can be found for example in [Collet, 2004] and [Pedersen, 2004].

In the present paper, beside the `Sim` algorithm, we will use a simple tree traversal algorithm —denoted `Enu`— to enumerate all configurations of mines in the fringe which are compatible with values in the boundary. Note that, for each such configuration (/solution) in the fringe, the number of possible mine configurations in the set of unsensed cells U is $C(m', |U|)$ (number of combinations/choices of m' elements among $|U|$) where m' is the number of mines in U. This makes it possible to compute each cell's probability to contain a mine.

Approximate Inference Because of the complexity of the inference problem, solving it exactly may not be feasible in a reasonable amount of time. Various strategies are then possible. First, one may interrupt the inference process as soon as a safe cell is identified. This can encourage searching for information regarding individual fringe cells, but possibly limiting the time for a single search.

	1	2	3	4	5	6	7	8
1	.	.	.	.	.	.	.	.
2	?	?	s	.	.	.	.	.
3	1	2	s	.	.	.	.	.
4		1	M	s	s	?	.	.
5		1	1	1	2	?	.	.
6					2	?	.	.
7				1	2	?	.	.
8	c			1	M	?	.	.

Figure 2: The same example as in Fig. 1 where some simple deductions have been performed indicating some fringe cells which are certainly mined (M) or safe (s).

With incomplete searches, one cannot be sure about the status of a cell. Yet, the number of solutions where c is mined or safe can give an approximation of $p_m(c)$:

$$\hat{p}_m(c) = \frac{\#(\text{cfgs where } c \text{ is mined}) + 1}{\#(\text{cfgs visited}) + 2},$$

where we use a Laplace estimate to avoid assuming that a cell is mined or safe unless the search was complete.

A particular approach is that of Castillo and Wrobel [2003], who base their inference phase not on a search algorithm, but on rules learned to classify cells. The resulting rules turn out to be competitive with John Ramsdell's PGMS program. One could relate this approach to CSP solvers learning pertinent rules on-line, or to learning control rules for CSP-based planning [Huang et al., 2000].

2.2.3 Inference in a Graphical Model

Given that we ideally want to know each cell's probability to contain a mine, and because of the graphical structure of the minefield, a very natural idea is to turn to probabilistic inference and more precisely to graphical models. This is what Vomlelova and Vomlel [2009] have done by employing Bayesian Networks.

Yet, they implicitly make the assumption that there is no global constraint, i.e., that mines are sampled independently in each cell (possibly with different distributions), or —more generally— that a cell's probability to contain a mine depends only on the presence of mines in adjacent cells. In the contrary, usual rules assume that the total number of mines is given, which implies that all cells are interdependent random variables.

2.2.4 Some Decision Rules

Let us first remind that our primary objective is to maximize the success probability, and that we do not attempt to minimize the number of actions before succeeding. Thus, as already mentioned, a first obvious decision rule is to pick any safe (uncovered) cell when one exists. This guarantees acquiring more information without any risk. In absence of safe cells (noting that new safe cells may appear after a move), other rules may be applied —depending on available inferred information— such as:

- **random:** picking any "possibly empty" cell at random;
- **lowProb:** picking a cell c minimizing the probability $p_m(c)$ (so as to maximize the probability to survive one more time step);
- **corner:** picking a possibly empty cell in a corner of the grid, because it has less chances to have an adjacent mine (because it has less adjacent cells), and thus more chances to contain a 0 and lead to an expansion to neighboring cells.

Of course, some rules can be combined: one may for example prefer cells in corners among low mine-probability cells.

Castillo and Wrobel [2003] describe two programs that autonomously play Minesweeper: John Ramsdell's PGMS, and their own player, which performs an inference based not on a CSP solver, but on rules learned to decide which cells are safe or are mines. Both algorithms exhibit very similar behaviors, and obtain 60% of victories on 10 000 games on 8×8 grids with 10 mines.

To our knowledge, little has been done to improve decisions. It is common to describe a game playing algorithm essentially through its inference process without detailing the decision rules. The purpose of this paper is thus precisely to explore this open area of automatic Minesweeper playing.

Assistants Note that a number of "intelligent" programs just serve as assistants, providing the result of their inference to a human player —visually indicating each cell's probability to contain a mine $p_m(c)$— but not proposing any decision. This is for example the case in [Collet, 2004, Bayer et al., 2006]. Such programs can only be evaluated based on their need for resources (mostly their computation time) and on their accuracy (if not exact).

2.3 POMDP View

POMDPs are usually defined [Monahan, 1982, Cassandra, 1998] by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, r, b_0 \rangle$ where, at any time step, the system being in some state $s \in \mathcal{S}$ (the *state space*), the agent performs an action $a \in \mathcal{A}$ (the *action space*) that results in (1) a transition to a state s' according to the *transition function* $T(s, a, s') = Pr(s'|s, a)$, (2) an observation $o \in \mathcal{O}$ (the *observation space*) according to the *observation function* $O(s', a, o) = Pr(o|s', a)$ and (3) a scalar *reward* $r(s, a)$. b_0 is the initial probability distribution over states. Unless stated otherwise, the sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ are finite.

In this setting, the problem is for the agent to find a decision *policy* π choosing, at each time step, the best action based on its past observations and actions so as to maximize its future gain (which can be measured for example through the total accumulated reward). Compared to classical deterministic planning, the agent has to face the difficulty of accounting for a system not only with uncertain dynamics but also whose current state is imperfectly known.

The agent typically reasons about the hidden state of the system using a *belief state* $b \in \mathcal{B} = \Pi(\mathcal{S})$ (the set of probability distributions over $\mathcal{S}$) using the following Bayesian update formula when performing action a and observing o:

$$b^{a,o}(s') = \frac{O(s', a, o)}{Pr(o|a, b)} \sum_{s \in \mathcal{S}} T(s, a, s') b(s),$$

where $Pr(o|a, b) = \sum_{s,s'' \in \mathcal{S}} O(s'', a, o) T(s, a, s'') b(s)$. Using belief states, a POMDP can be rewritten as an MDP over the belief space, or *belief MDP*, $\langle \mathcal{B}, \mathcal{A}, \mathcal{T}, \rho \rangle$, where the new transition and reward functions are both defined over $\mathcal{B} \times \mathcal{A} \times \mathcal{B}$. With this reformulation, a number of

theoretical results about MDPs can be extended, such as the existence of a deterministic policy that is optimal. An issue is that this belief MDP is defined over a continuous —and thus infinite— belief space.

For a finite horizon[4] $T > 0$ the objective is to find a policy verifying $\pi^* = \arg\max_{\pi \in \mathcal{A}^{\mathcal{B}}} J^\pi(b_0)$ with

$$J^\pi(b_0) = E\left[\sum_{t=0}^{T-1} \gamma^t r_t \middle| b_0, \pi\right],$$

where b_0 is the initial belief state, r_t the reward obtained at time step t, and $\gamma \in (0,1)$ a discount factor. Bellman's principle of optimality [Bellman, 1954] lets us compute this function recursively through the *value function*

$$V_n(b) = \max_{a \in \mathcal{A}} \left[\rho(b,a) + \gamma \sum_{b' \in \mathcal{B}} \phi(b,a,b') V_{n-1}(b')\right],$$

where, for all $b \in \mathcal{B}$, $V_0(b) = 0$, and $J^\pi(b) = V_{n=T}(b)$.

A number of algorithms exploit the fact that $V_n(b)$ is a piecewise linear and convex function (PWLC). This allows for direct computations or approximations where the target function is the upper envelope of a set of hyperplanes. This led to algorithms performing exact updates like *Batch Enumeration* [Monahan, 1982], *Witness* or *Incremental Pruning* (IP) [Cassandra, 1998], but also approximate ones as in *Point-Based Value Iteration* (PBVI) [Pineau et al., 2006], *Heuristic Search Value Iteration* (HSVI) [Smith and Simmons, 2004], PERSEUS [Spaan and Vlassis, 2005] or SARSOP [Kurniawati et al., 2008].

If not relying on PWLC functions, one can also solve a POMDP as an MDP on a continuous state space, for example with tree search algorithms —which allow for online planning [Ross et al., 2008]— or with RTDP-bel, a variant of dynamic programming that continuously focuses on relevant parts of the state space [Bonet and Geffner, 2009].

Let us also mention that some algorithms —such as Symbolic HSVI [Sim et al., 2008]— can exploit the structure of a factored POMDP, i.e., a POMDP in which the state and/or the observation is described through multiple variables.

2.3.1 POMDP Model

The game of Minesweeper can be naturally modeled as a POMDP $\langle S, S_g, A, T, R, O, \Omega, b_0 \rangle$ where:

- the set of states S is made of
  - normal states, described by the possible minefields $M \in \mathcal{M}_{W \times H, m}$ and the updated knowledge matrix K (indicating which cells have been uncovered), plus
  - two special states: (i) "init" (s_{init}), which corresponds to the state before the first action (prior to the minefield's generation), and (ii) "failure" (s_{failure}) which is absorbing (any action will leave us there);
- goal (/terminal) states (S_g) are those where K has only m cells with value -1 (which are "success" states), plus the failure state; as the failure state, all of them are absorbing;
- actions in A and the transition function T are as described in the previous section (plus a "fake" action for use in absorbing state);

[4] In practice we consider an infinite horizon.

- $R(s, a, s') = 0$ in all transitions except when transiting from $S \setminus S_g$ to S_g (what can happen only once), where the $R(s, a, s') = 1$;
- observations in O correspond to the possible knowledge matrices K;
- the observation function Ω updates the knowledge matrix K according to the last action;
- b_0 is a probability distribution putting all the weight on the init state: $b_0(s_{\text{init}}) = 1$.

One could consider a variant with a reward function defined to be 0 everywhere except when transiting to the failure state, where $R(s, a, s') = -1$. This would lead us to minimizing the probability of failure instead of maximizing the probability of success, which are equivalent objectives. In fact, the "success" reward can be any positive value, and the "failure" reward any negative value, and at least one of them at least to be non-zero. What should not be done is to assign any kind of reward to other transitions. Indeed, assigning unit costs to each action would lead to making a compromise between maximizing the probability to reach the goal and minimizing the expected plan length.

As can be observed, this POMDP has a number of particular features, some of them that could be exploited:

1. it has terminal states (S_g), which will be reached in finite time with probability 1 (the horizon is then *indefinite*); algorithms such as RTDP-bel could be used with a formalization with only negative rewards (when reaching the failure state);
2. as long as no terminal state is reached, one can only accumulate new information in K, and thus in the belief state b; if we represent the possible evolutions of the belief state, we thus get an AND-OR directed acyclic graph (where AND nodes correspond to decision of the player, and OR nodes correspond to chance), the acyclicity being due to the accumulation of information;
3. obviously, the (indefinite) time horizon is upper-bounded by $W \cdot H - m$;
4. when a safe cell is found, one can constrain the solver to pick it; this will reduce the search space;
5. except when such obvious choices are possible, the number of possible actions is typically large;
6. part of the state (K) is fully observable, so that we are facing a *mixed observability Markov decision process* (MOMDP) [Ong et al., 2009, Araya-López et al., 2010]; this allows for dramatic speed-ups by reducing the dimensionality of the belief space to consider, as observed in MO-IP or MO-SARSOP.

These various observations show that various algorithms could be more efficient than a classical point-based approach: RTDP-bel because we are dealing with an shortest-path problem, MO-SARSOP because of the mixed-observability, or maybe a Monte-Carlo Tree Search algorithm with limited horizon. Note that each of these approaches relies on different features of the problem. None of them exploits multiple features. Plus, in any case the size of the search space remains too large and the transition function is too complex to compute.

Table 1: Comparison of important features in 3 similar games

game	Minesweeper	Hangman	Mastermind
hidden state	minefield M	word ω	pattern of colors p
actions	try hidden cell c	try letter λ	try pattern of colors p'
observation	K updated in $N_M(c)$	occurrences of λ in ω	"compatibility" between p and p'
failure	if c mined	if too many attempts	never
objective	$\max Pr(\text{win})$	$\max Pr(\text{win})$	$\min t_{\text{win}}$

2.3.2 MDP Approaches

Nakov and Wei [2003] model MineSweeper as a —fully observable— Markov Decision Process. They in fact by-pass the POMDP view to directly consider a bMDP. Yet, they do so without referring to the notion of belief, considering instead the state to be the knowledge matrix K, which is valid since K is equivalent to the belief (K is the union of past observations, which is the only information needed to compute the belief).

Nakov and Wei [2003] solve this MDP using Value Iteration, and exploiting symmetries and sure moves. Yet, they do not go beyond a 4×4 grid because of the combinatorial explosion of the state space.

Sebag and Teytaud [2012] also address MineSweeper as an MDP, but address the scalability issue by using a Monte-Carlo Tree Search which can exploit heuristic rules. This approach avoids enumerating the complete state space, so that the authors can experiment with grids of size up to 5×5 (which already is a significant improvement if you consider the combinatorial explosion of the state space).

3 Approach: How to Safely Gain Information?

Considering how (PO)MDP approaches suffer from the combinatorial explosion of the state space, we have decided to search for more specific —but heuristic— solutions by looking more closely at the game of Minesweeper.

3.1 Looking back at the Game and its Objective

The primary objective of the Minesweeper player is to maximize the probability to be in a success state when the game ends. Up to now, we have characterized success states as states in which all safe cells are uncovered in K (only m cells marked with a -1 remain). Then, assuming that we constrain our policies to pick any provably safe cell, any state in which we provably know where all mines are can also be considered to be a success state.

Another important observation is that Minesweeper [wikipedia, 2011c] resembles Hangman [wikipedia, 2011a] or Mastermind [wikipedia, 2011b] in that the objective is to discover a hidden state by performing sensing actions. Many other games could be added to the list, but we hereby focus on the three most popular ones. Table 1 outlines the similarities and differences between those three games by comparing some key aspects. Minesweeper and Hangman are the most similar games in the sense that both have failure states to avoid —hence the identical objective of maximizing the success probability— while Mastermind is about minimizing the number of guesses before finding the solution (which will necessarily be found at some point).

Mastermind seems to have been the most studied of these games [wikipedia, 2011b, Weisstein, 2011]. Various algorithms have been proposed (including optimal ones). As explained in [Kooi, 2005], in the absence of possible traps (failure states), one essentially wants to acquire information

as quickly as possible, which depends on the possible outcomes of an action a. Indeed, given the current set P of possible patterns (assumed equiprobable), an action a induces a partition of P into one subset $P(a, o)$ per possible next observation o. Thus, among possible immediate actions, one can choose for example the action a:

- minimizing $\max_o |P(a, o)|$ (the largest possible set of remaining candidate patterns); this is the Worst-Case Strategy used by Knuth [1976–77]; or
- maximizing the expected entropy.

Similar ideas have been discussed to play Hangman as a codebreaker —or to trick the codebreaker with hard-to-guess words— for example by McLoone [2010].

3.2 Heuristic Rules

As can be noted, none of the greedy algorithms —i.e., algorithms not relying on exploring possible futures with a horizon greater than 1— developed for these games is provably optimal. In the same vein, we here propose heuristics based on the idea of maximizing the gain of information.

In Minesweeper, estimating the expected gain of information following a given action a would require reasoning about all possible outcomes, i.e., computing, for each possible observation, some quantity depending on the possible minefield configurations. Another option is to consider that actions with the largest expected uncovered area are among the actions providing the most information. Of course, they may not be the safest ones.

A question is then how to identify such actions maximizing the expected uncovered area. We are here facing a percolation problem: how does a fluid flow through our minefield? Yet, percolation as studied in complex systems typically concerns infinite environments, such as the infinite Minesweeper examined by Mossel [2002]. In our finite setting, two exact approaches would be:

1. for each candidate cell c, compute the uncovered area for each possible minefield configuration and then derive the expected uncovered surface for c;
2. take the opposite point of view and count, for each possible uncovered area (not accounting for the numbers appearing in K), how many minefield configurations are compatible with it.

In both cases, one could exploit the fact that some regions are unreachable from c. But, in the latter approach, the choice of an uncovered area leads us to a new inference problem requiring again to reason mainly about fringe cells (cells adjacent to uncovered cells) and thus save computation time.

Still, these two processes would require a lot of computational effort. We instead consider three intuitions:

- cells *close to the "frame"* (the border of the grid) have less adjacent cells, and thus higher chances to be zero-valued, so that these adjacent cells are in turn more likely to be uncovered;
- cells *far from both the boundary and the frame* have higher chances to uncover a wide area because all evolutions in all directions are possible;
- cells *close to the boundary* may not be likely to uncover a large area —because cells in the fringe maybe mined— but are likely to bring more information about the suspicious cells in the fringe.

These intuitions may contradict each other, and it is possible that which intuition is right may depend for example on the density of mines. We will thus experiment with all three, or more precisely with the following cases where we prefer the cell c minimizing:

[Aw] $d(c, boundary)$: picking a cell as far from the boundary as possible;

[Awf] $\min(d(c, boundary), d(c, frame))$: picking a cell as far from the boundary and the frame as possible;

[Cl] $-d(c, boundary)$: picking a cell close to the boundary; and

[Clf] $-\min(d(c, boundary), d(c, frame))$: picking a cell close either to the boundary or to the frame.

For the boundary, we use the Manhattan distance to previously uncovered cells in K. Let $d_{\max}$ be this distance. Computing this Manhattan distance for each covered cell can be done using a breadth-first search, hence with a complexity $O(|U(C)|)$ (see Algorithm 1).

Algorithm 1: Computing all covered cells' Manhattan distance to uncovered cells.

```
   /* Note: α'(c) is the set of c's 4-adjacent cells.                    */
 1 foreach c do d(c) ← −1 /* Initialize all distances to −1.            */
 2 S ← φ(C) /* Source set initialized to boundary cells.                 */
 3 foreach c ∈ S do d(c) ← 0 /* Boundary cells are at distance 0.        */
 4 repeat
 5 |  D ← ∅ /* Destination set initially empty.                          */
 6 |  foreach c ∈ S do
 7 |  |  foreach c' ∈ α'(c) ∩ U(C) do
 8 |  |  |  if d(c') = −1 then
 9 |  |  |  |  D ← D ∪ {c'}
10 |  |  |  |  d(c') ← d(c) + 1
11 |  S ← D
12 until S = ∅
```

As can be observed, one may want to act close to the boundary —where useful information may be gained to complement what is already known about some cells— but not in the fringe if the probability to find a mine is too high. We will thus combine each of the four aforementioned "distance rules" with a "low mine probability" rule in one of two ways: (i) among the lowest mine probability cells, pick one optimizing the distance rule, and (ii) among the cells optimizing the distance rule, pick one with minimum mine probability.

First Move A particular question is which cell to probe first. This is essentially a particular case of the general probing problem, where (i) no boundary or fringe exists, (ii) the cell to be probed will be safe, and (iii) all other cells will have the same probability to be mined. The heuristic rules discussed previously can then be reduced to two rules, to which we add a third one for our experiments:

- probe randomly, e.g., $(rand(1, W), rand(1, H))$;
- probe a corner, e.g., $(1, 1)$;

- probe in the middle, e.g., $(\lfloor W \rfloor, \lfloor H \rfloor)$.

Note that first clicks have been already looked at in the context of human Minesweeper. Kostka [2006] has for example experimentally studied on windows minesweeper:

- the probability to open up a space (i.e., the probability for the first cell not to be adjacent to a mine), showing that —as expected from the theory— the more adjacent cells, the lower this probability (but with a bias in this particular implementation), and
- in case a space is opened, its expected surface increases when one moves towards the center of the grid.

As can be noted, these two phenomena taken independently would lead to contradictory decision rules.

4 Experiments

4.1 Implementation Details

The strategies that we have implemented and experimented with are based on the following building block algorithms:

..... [*first-move* building blocks] .....

`r/c/m` picks the first cell `r`andomly, in a `c`orner, or in the `m`iddle;

..... [*non-decisive* building blocks] .....

`Sim` applies simple rules allowing quick identification of safe or mined cells:

1. flag obvious mines,
2. pick any uncovered cell around a 0-valued cell,
3. if a k-valued cell is surrounded by k flagged mines, other adjacent cells are safe;

`Enu` enumerates all solutions in the fringe (and computes each cell's c probability $P_m(c)$); this algorithm allows to flag any provably mined cell or pick any provably safe cell; it is interrupted if its execution lasts more than one second;

..... [*tie-breaking* building blocks] .....

`Ran` picks a cell randomly;

`Lo` prefers cells c minimizing $P_m(c)$;

`Aw` prefers cells c away from the boundary;

`Awf` prefers cells c away from the boundary and the fringe;

`Cl` prefers cells c close to the boundary;

`Clf` prefers cells c close the boundary or the fringe.

Thus, "`cSimEnuRan`" denotes a strategy that (i) for the first move, probes a corner, then —for each subsequent move— uses (ii) the simple rules to identify safe or mined cells, before (iii) enumerating valid fringe configurations, or (iv) acts randomly if no safe choice is possible. `Lo` can be combined with `Aw`, `Awf`, `Cl` and `Clf` as described previously. In each case, one moves to the next building block algorithm if the previous one failed to find a cell to play.

Table 2: "Official" (and alternate) parameters for the three reference game levels

	Beginner	(alt.)	Intermediate	(alt.)	Expert
Grid Size	9×9	(8×8)	16×16	(13×15)	16×30
# mines	10	(10)	40	(40)	99
Density	12.3%	(15.6%)	15.6%	(20.5%)	20.6%

Table 3: State of the art results (success rates)

	Beginner-alt	Intermediate-alt	Expert
Single Point	?	?	?
Equation	0.71	0.36	0.26
Mio	0.71	0.36	0.26

4.2 Benchmark Settings

We experiment with the three reference game levels described by their grid sizes and number of mines in Table 2. The game difficulty increases with the mine density (percentage of mined cells) or the grid-size. When running our own program, for each game level and each algorithm, we randomly generate the same set of minefields to play with (using the same random seed).

All experiments are conducted on a single core of an i5 CPU at 2.53 GHz. But note that timing information are essentially indicative as limited effort has been put into optimizing the enumeration algorithm.

Our strategies can be compared with three reference approaches:

- *Single Point* uses simple rules to identify many mined and safe cells (no use of a complete CSP solver), but otherwise acts randomly; using our naming scheme, it is denoted `SimRan`;
- *Equation* derives integer linear equations, starting from the ones required to describe the inference problem, and combining some of them to derive new equations; it then repeatedly applies rules based on these equations to identify safe or mined cells; when this process stops, each cell's probability to be mined is heuristically approximated by looking at the values around it, and a cell with minimum mine probability is probed;
- *Mio* is similar to *Equation*, but uses rules learned —under the form of clauses— through experiments; the resulting rules are comparable to those from *Equation*, and similar experimental results were obtained.

Table 3 shows success probabilities obtained by these approaches as reported in `http://www.ccs.neu.edu/home/ramsdell/pgms/games.html`. For Single Point (`SimRan`), see next section.

4.3 Results

In this section, we (i) compare a set of various strategies on a "beginner" grid, then, using a selection of three algorithms, we (ii) study the impact of the first move, and (iii) look at further results at "intermediate" and "expert" levels.

4.3.1 Comparing Various Strategies

Table 4 lists the various strategies we experimented with (here with a random first probe), showing the average success rate and the total running time for 10 000 random beginner minefields.

Table 4: Preliminary results: for each algorithm are given the average success probability and the total computation time (in milliseconds) for 10 000 random beginner minefields, and with a random first probe

	P_{win}	Total time
rSimRan	0.67	5747
rEnuRan	0.81	480041
rSimEnuRan	0.81	58032
rSimEnuLo	0.85	30901
rSimEnuAwRan	0.84	10995
rSimEnuAwLo	0.87	15097
rSimEnuClLo	0.85	31088
rSimEnuAwLof	0.82	282397
rSimEnuClLof	0.85	30724
rSimEnuLoAw	0.87	14560
rSimEnuLoCl	0.85	9419
rSimEnuLoAwf	0.83	237018
rSimEnuLoClf	0.87	8266

Table 5: Additional statistics: for each algorithm are given the average success probability, the average probability of a timeout of the enumeration algorithm, plus statistics (minimum/median–average/maximum) about the average duration of a game, and the number of CSPs that have been solved

	P_w	P_t	GameDuration	nbSolvedCSPs	totNbSolutions
SimRan-c9-l9-m10-i(-1,-1)	0.670	0.0000	(0.00/0.00– 0.43/ 25.00)	(0.00/ 0.00– 0.00/ 0.00)	(0.00/0.00– 0.00/ 0.00)
EnuRan-c9-l9-m10-i(-1,-1)	0.808	0.0000	(0.00/3.00–47.78/30653.00)	(2.00/71.00–59.65/71.00)	(1.00/5.30– 507.83/565612.31)
SimEnuRan-c9-l9-m10-i(-1,-1)	0.808	0.0003	(0.00/1.00– 5.64/ 4425.00)	(0.00/ 1.00– 2.00/20.00)	(0.00/8.00–1021.38/636313.75)
SimEnuLo-c9-l9-m10-i(-1,-1)	0.849	0.0000	(0.00/1.00– 2.94/ 1338.00)	(0.00/ 1.00– 2.12/21.00)	(0.00/8.00– 608.28/330588.34)
SimEnuAwRan-c9-l9-m10-i(-1,-1)	0.845	0.0000	(0.00/1.00– 0.94/ 1094.00)	(0.00/ 1.00– 1.81/20.00)	(0.00/8.00– 95.53/140537.00)
SimEnuAwLo-c9-l9-m10-i(-1,-1)	0.871	0.0001	(0.00/1.00– 1.35/ 4497.00)	(0.00/ 1.00– 1.81/21.00)	(0.00/7.50– 154.86/567223.44)
SimEnuClLo-c9-l9-m10-i(-1,-1)	0.849	0.0000	(0.00/1.00– 2.95/ 1341.00)	(0.00/ 1.00– 2.12/21.00)	(0.00/8.00– 608.28/330588.34)
SimEnuAwLof-c9-l9-m10-i(-1,-1)	0.823	0.0066	(0.00/1.00–28.07/ 6855.00)	(0.00/ 1.00– 2.08/21.00)	(0.00/8.00–6791.83/938017.81)
SimEnuClLof-c9-l9-m10-i(-1,-1)	0.849	0.0000	(0.00/1.00– 2.91/ 1344.00)	(0.00/ 1.00– 2.12/21.00)	(0.00/8.00– 608.28/330588.34)
SimEnuLoAw-c9-l9-m10-i(-1,-1)	0.873	0.0001	(0.00/1.00– 1.30/ 4514.00)	(0.00/ 1.00– 1.87/22.00)	(0.00/7.50– 132.21/539675.69)
SimEnuLoCl-c9-l9-m10-i(-1,-1)	0.847	0.0000	(0.00/1.00– 0.78/ 65.00)	(0.00/ 1.00– 2.28/24.00)	(0.00/7.83– 62.20/ 19763.80)
SimEnuLoAwf-c9-l9-m10-i(-1,-1)	0.830	0.0033	(0.00/1.00–23.54/ 4871.00)	(0.00/ 1.00– 2.13/25.00)	(0.00/8.00–5789.86/856261.69)
SimEnuLoClf-c9-l9-m10-i(-1,-1)	0.870	0.0000	(0.00/1.00– 0.67/ 32.00)	(0.00/ 1.00– 2.18/24.00)	(0.00/6.00– 17.60/ 1690.50)

The first three lines show that the `sim`ple rules alone are quite efficient at this level of difficulty (67% of success). Plus, they are fast and dramatically speed up the `enu`meration algorithm. Probing safe cells or acting randomly in absence of safe cells allows attaining a score of 81% (65% in the alternate beginner level, which is close to the state of the art).

The remaining lines show that our decision rules improve over the state of the art —with 82%–87% of successes. These results, and the total running times, tend to show that probing cells "in the middle" is not very informative, leading to longer, more complex (harder enumeration problems), and less successful games. Plus, it seems preferable to make a choice among low mine probability cells.

Tables 5 and 6 provide more statistical results regarding these experiments, illustrating in particular the following facts: (i) most problems are solved very quickly, (ii) picking cells in the middle leads to problems with more variables (more cells in the fringe), and (iii) problems with more variables are harder to solve. Also, preferring cells away from the fringe leads to the largest maximum problem sizes, with a notably higher probability for the enumeration algorithm to time out.

In the remainder of our experiments, we have focused on 5 strategies: *Single Point* (`SimRan`),

Table 6: Additional statistics: for each algorithm are given statistics (minimum/median–average/maximum) about the average number of cells in the fringe of a game's CSPs, the maximum number of cells in the fringe of a game's CSPs, and the number of uncovered cells at the end of the game

	nbCellsInFringe	maxCellsInFringe	nbUncoveredCells	maxNbSolutions
SimRan-c9-l9-m10-i(-1,-1)	(0.00/ 0.00– 0.00/ 0.00)	(0.00/ 0.00– 0.00/ 0.00)	(1.00/71.00–55.94/71.00)	(0.00/ 0.00– 0.00/ 0.00)
EnuRan-c9-l9-m10-i(-1,-1)	(1.50/12.68–12.79/31.31)	(3.00/21.00–21.73/56.00)	(1.00/71.00–59.46/71.00)	(1.00/28.00– 4913.07/2617720.00)
SimEnuRan-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 7.85/38.17)	(0.00/ 8.00–10.16/54.00)	(1.00/71.00–59.46/71.00)	(0.00/ 8.00– 4243.17/2202200.00)
SimEnuLo-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 7.54/31.50)	(0.00/ 8.00– 9.57/48.00)	(1.00/71.00–62.06/71.00)	(0.00/ 8.00– 2533.38/1598464.00)
SimEnuAwRan-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 7.64/30.83)	(0.00/ 8.00– 9.21/50.00)	(1.00/71.00–62.69/71.00)	(0.00/ 8.00– 452.53/1209600.00)
SimEnuAwLo-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 6.93/33.15)	(0.00/ 8.00– 8.46/59.00)	(1.00/71.00–63.63/71.00)	(0.00/ 8.00– 743.60/2153663.00)
SimEnuClLo-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 7.54/31.50)	(0.00/ 8.00– 9.57/48.00)	(1.00/71.00–62.06/71.00)	(0.00/ 8.00– 2533.38/1598464.00)
SimEnuAwLof-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 8.87/48.36)	(0.00/ 8.00–12.17/72.00)	(1.00/71.00–60.40/71.00)	(0.00/ 8.00–27068.65/3231667.00)
SimEnuClLof-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 7.54/31.50)	(0.00/ 8.00– 9.57/48.00)	(1.00/71.00–62.06/71.00)	(0.00/ 8.00– 2533.38/1598464.00)
SimEnuLoAw-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 6.96/33.15)	(0.00/ 8.00– 8.47/59.00)	(1.00/71.00–63.77/71.00)	(0.00/ 8.00– 599.82/2056349.00)
SimEnuLoCl-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 7.17/26.28)	(0.00/ 8.00– 8.77/36.00)	(1.00/71.00–61.93/71.00)	(0.00/ 8.00– 167.05/ 59241.00)
SimEnuLoAwf-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 8.62/35.30)	(0.00/ 8.00–11.56/54.00)	(1.00/71.00–60.87/71.00)	(0.00/ 8.00–22896.46/2641341.00)
SimEnuLoClf-c9-l9-m10-i(-1,-1)	(0.00/ 8.00– 6.95/23.25)	(0.00/ 8.00– 8.29/33.00)	(1.00/71.00–63.49/71.00)	(0.00/ 8.00– 33.46/ 6384.00)

Table 7: Main Results: for each algorithm are given the average success probability and the total computation time (in milliseconds) for 10 000 games

	Random		Corner		Middle	
SimRan	0.67	(5686)	0.70	(6090)	0.66	(5692)
SimEnuAwLo	0.87	(15150)	0.91	(8563)	0.85	(9497)
SimEnuLoAw	0.87	(14820)	0.91	(8515)	0.85	(9395)
SimEnuLoCl	0.85	(9450)	0.90	(8437)	0.82	(10443)
SimEnuLoClf	0.87	(8300)	0.90	(8616)	0.85	(8571)

and 4 strategies that have a success rate above 85% in these first experiments.

4.3.2 Impact of the First Move

The figures in Table 7 clearly demonstrate that —at the beginner level— one should prefer first moves in the corner. This is consistent with previous results, where clicking "in the middle" proved to be less efficient. Plus, again the total resolution time correlates with the success rate (among `SimEnu*` strategies).

Tables 8 and 9 provide more detailed results confirming this analysis. Note that experiments in the same conditions as in Section 4.3.1 —here, with a random first move— have been re-executed, producing very similar, but not identical, results.

4.3.3 Results on More Complex Grids

A more important question is whether the 4 selected strategies —now with a first move in a corner— scale well to more difficult games. They do indeed, as can be observed in Tables 10 and 11. Table 11 shows success rates notably above the state of the art for the 3 strategies putting more importance on safety (low mine probability): 50% (vs. 36%) at the intermediate level, and 37–38% (vs. 26%) at the expert level. `SimEnuAwLo` is not as robust, but remains above the state of the art.

Again, more detailed statistics can be found in Tables 12 and 13.

Figure 3 shows, for `SimRan` and `SimEnuLoClf`, which proportion of the probed cells come from each building block algorithm. Note that these are only proportions within all moves of each strategy. Higher values for one strategy do not mean that this strategy has performed an action more often than another strategy.

Table 8: Additional statistics: for each algorithm are given the average success probability, the average probability of a timeout of the enumeration algorithm, plus statistics (minimum/median–average/maximum) about the average duration of a game, and the number of CSPs that have been solved

	P_w	$P_!$	GameDuration	nbSolvedCSPs	totNbSolutions
SimRan-c9-l9-m10-i(-1,-1)	0.670	0.0000	(0.00/0.00–0.42/ 25.00)	(0.00/0.00–0.00/ 0.00)	(0.00/0.00– 0.00/ 0.00)
SimRan-c9-l9-m10-i(0,0)	0.702	0.0000	(0.00/0.00–0.45/ 23.00)	(0.00/0.00–0.00/ 0.00)	(0.00/0.00– 0.00/ 0.00)
SimRan-c9-l9-m10-i(4,4)	0.663	0.0000	(0.00/0.00–0.41/ 20.00)	(0.00/0.00–0.00/ 0.00)	(0.00/0.00– 0.00/ 0.00)
SimEnuAwLo-c9-l9-m10-i(-1,-1)	0.871	0.0001	(0.00/1.00–1.36/4473.00)	(0.00/1.00–1.81/21.00)	(0.00/7.50–157.70/595599.25)
SimEnuAwLo-c9-l9-m10-i(0,0)	0.911	0.0000	(0.00/1.00–0.70/ 328.00)	(0.00/1.00–1.70/21.00)	(0.00/2.00– 20.29/106824.00)
SimEnuAwLo-c9-l9-m10-i(4,4)	0.851	0.0000	(0.00/1.00–0.80/ 242.00)	(0.00/1.00–1.84/21.00)	(0.00/8.00– 64.84/ 75824.00)
SimEnuLoAw-c9-l9-m10-i(-1,-1)	0.873	0.0001	(0.00/1.00–1.33/4522.00)	(0.00/1.00–1.87/22.00)	(0.00/7.50–131.46/532189.25)
SimEnuLoAw-c9-l9-m10-i(0,0)	0.914	0.0000	(0.00/1.00–0.69/ 43.00)	(0.00/1.00–1.82/21.00)	(0.00/2.00– 8.22/ 7406.38)
SimEnuLoAw-c9-l9-m10-i(4,4)	0.853	0.0000	(0.00/1.00–0.79/ 252.00)	(0.00/1.00–1.92/21.00)	(0.00/8.00– 54.74/ 75824.00)
SimEnuLoCl-c9-l9-m10-i(-1,-1)	0.847	0.0000	(0.00/1.00–0.78/ 63.00)	(0.00/1.00–2.28/24.00)	(0.00/7.83– 62.20/ 19763.80)
SimEnuLoCl-c9-l9-m10-i(0,0)	0.902	0.0000	(0.00/1.00–0.68/ 39.00)	(0.00/1.00–2.29/28.00)	(0.00/2.50– 3.91/ 390.89)
SimEnuLoCl-c9-l9-m10-i(4,4)	0.820	0.0000	(0.00/1.00–0.88/ 54.00)	(0.00/1.00–2.36/23.00)	(0.00/9.00– 94.07/ 10363.00)
SimEnuLoClf-c9-l9-m10-i(-1,-1)	0.870	0.0000	(0.00/1.00–0.67/ 22.00)	(0.00/1.00–2.18/24.00)	(0.00/6.00– 17.60/ 1690.50)
SimEnuLoClf-c9-l9-m10-i(0,0)	0.902	0.0000	(0.00/1.00–0.70/ 39.00)	(0.00/1.00–2.29/28.00)	(0.00/2.50– 3.92/ 401.00)
SimEnuLoClf-c9-l9-m10-i(4,4)	0.853	0.0000	(0.00/1.00–0.69/ 34.00)	(0.00/1.00–2.18/23.00)	(0.00/8.00– 22.80/ 920.45)

Table 9: Additional statistics: for each algorithm are given statistics (minimum/median–average/maximum) about the average number of cells in the fringe of a game's CSPs, the maximum number of cells in the fringe of a game's CSPs, and the number of uncovered cells at the end of the game

	nbCellsInFringe	maxCellsInFringe	nbUncoveredCells	maxNbSolutions
SimRan-c9-l9-m10-i(-1,-1)	(0.00/0.00–0.00/ 0.00)	(0.00/0.00– 0.00/ 0.00)	(1.00/71.00–55.94/71.00)	(0.00/ 0.00– 0.00/ 0.00)
SimRan-c9-l9-m10-i(0,0)	(0.00/0.00–0.00/ 0.00)	(0.00/0.00– 0.00/ 0.00)	(1.00/71.00–58.54/71.00)	(0.00/ 0.00– 0.00/ 0.00)
SimRan-c9-l9-m10-i(4,4)	(0.00/0.00–0.00/ 0.00)	(0.00/0.00– 0.00/ 0.00)	(1.00/71.00–55.53/71.00)	(0.00/ 0.00– 0.00/ 0.00)
SimEnuAwLo-c9-l9-m10-i(-1,-1)	(0.00/8.00–6.93/33.15)	(0.00/8.00– 8.46/59.00)	(1.00/71.00–63.63/71.00)	(0.00/ 8.00–753.33/2250987.00)
SimEnuAwLo-c9-l9-m10-i(0,0)	(0.00/4.00–4.77/23.93)	(0.00/4.00– 6.02/48.00)	(1.00/71.00–66.47/71.00)	(0.00/ 3.00–120.85/ 840000.00)
SimEnuAwLo-c9-l9-m10-i(4,4)	(0.00/8.00–7.46/26.00)	(0.00/8.00– 8.92/41.00)	(1.00/71.00–62.33/71.00)	(0.00/ 8.00–272.82/ 472500.00)
SimEnuLoAw-c9-l9-m10-i(-1,-1)	(0.00/8.00–6.96/33.15)	(0.00/8.00– 8.47/59.00)	(1.00/71.00–63.77/71.00)	(0.00/ 8.00–598.57/2043879.00)
SimEnuLoAw-c9-l9-m10-i(0,0)	(0.00/4.00–4.82/21.23)	(0.00/4.00– 6.02/38.00)	(1.00/71.00–66.65/71.00)	(0.00/ 3.00– 29.20/ 56700.00)
SimEnuLoAw-c9-l9-m10-i(4,4)	(0.00/8.00–7.49/26.20)	(0.00/8.00– 8.91/41.00)	(1.00/71.00–62.44/71.00)	(0.00/ 8.00–220.01/ 472500.00)
SimEnuLoCl-c9-l9-m10-i(-1,-1)	(0.00/8.00–7.17/26.28)	(0.00/8.00– 8.77/36.00)	(1.00/71.00–61.93/71.00)	(0.00/ 8.00–167.05/ 59241.00)
SimEnuLoCl-c9-l9-m10-i(0,0)	(0.00/4.50–4.95/20.46)	(0.00/6.00– 6.22/31.00)	(1.00/71.00–65.82/71.00)	(0.00/ 3.00– 7.26/ 1359.00)
SimEnuLoCl-c9-l9-m10-i(4,4)	(0.00/8.00–8.37/26.94)	(0.00/8.00–10.34/41.00)	(1.00/71.00–60.15/71.00)	(0.00/19.00–244.19/ 27720.00)
SimEnuLoClf-c9-l9-m10-i(-1,-1)	(0.00/8.00–6.95/23.25)	(0.00/8.00– 8.29/33.00)	(1.00/71.00–63.49/71.00)	(0.00/ 8.00– 33.46/ 6384.00)
SimEnuLoClf-c9-l9-m10-i(0,0)	(0.00/4.50–4.95/20.46)	(0.00/6.00– 6.23/31.00)	(1.00/71.00–65.82/71.00)	(0.00/ 3.00– 7.23/ 1638.00)
SimEnuLoClf-c9-l9-m10-i(4,4)	(0.00/8.00–7.59/26.00)	(0.00/8.00– 8.94/35.00)	(1.00/71.00–62.47/71.00)	(0.00/ 8.00– 43.08/ 5040.00)

Table 10: Main Results: for each algorithm are given the average success probability and the total computation time (in milliseconds) for 10 000 games

	Beginner		Intermediate		Expert	
cSimEnuAwLo	0.91	(8637)	0.75	(276514)	0.29	(4335034)
cSimEnuLoAw	0.91	(8398)	0.78	(121324)	0.37	(1109291)
cSimEnuLoCl	0.90	(8521)	0.76	(58035)	0.38	(361727)
cSimEnuLoClf	0.90	(8494)	0.76	(56129)	0.38	(378512)

Table 11: Same information as in Table 10, but for *alternate* grids (except in Expert mode, where the differences between numbers is due to a new set of experiments)

	Beginner		Intermediate		Expert	
Equation	0.71		0.36		0.26	
Mio	0.71		0.36		0.26	
cSimRan	0.46	(4757)	0.06	(10701)	0.01	(33399)
cSimEnuAwLo	0.80	(19363)	0.42	(1682675)	0.29	(4249418)
cSimEnuLoAw	0.81	(17031)	0.50	(358475)	0.37	(1082953)
cSimEnuLoCl	0.80	(7574)	0.50	(45822)	0.38	(374661)
cSimEnuLoClf	0.80	(7645)	0.50	(44974)	0.38	(369989)

Table 12: Additional statistics: for each algorithm are given the average success probability, the average probability of a timeout of the enumeration algorithm, plus statistics (minimum/median–average/maximum) about the average duration of a game, and the number of CSPs that have been solved

	P_w	P_1	GameDuration	nbSolvedCSPs	totNbSolutions
SimRan-c9-l9-m10-i(0,0)	0.702	0.0000	(0.00/ 0.00– 0.46/ 26.00)	(0.00/ 0.00– 0.00/ 0.00)	(0.00/ 0.00– 0.00/ 0.00)
SimRan-c16-l16-m40-i(0,0)	0.387	0.0000	(0.00/ 4.00– 3.34/ 85.00)	(0.00/ 0.00– 0.00/ 0.00)	(0.00/ 0.00– 0.00/ 0.00)
SimRan-c30-l16-m99-i(0,0)	0.009	0.0000	(0.00/ 0.00– 2.94/ 69.00)	(0.00/ 0.00– 0.00/ 0.00)	(0.00/ 0.00– 0.00/ 0.00)
SimEnuAwLo-c9-l9-m10-i(0,0)	0.911	0.0000	(0.00/ 1.00– 0.71/ 317.00)	(0.00/ 1.00– 1.70/21.00)	(0.00/ 2.00– 20.29/ 106824.00)
SimEnuAwLo-c16-l16-m40-i(0,0)	0.753	0.0073	(0.00/ 6.00– 27.42/10624.00)	(0.00/ 3.00– 3.58/33.00)	(0.00/ 5.00– 3391.40/1086678.75)
SimEnuAwLo-c30-l16-m99-i(0,0)	0.291	0.0793	(0.00/21.00–433.25/25462.00)	(0.00/ 7.00– 8.92/51.00)	(0.00/94.15–37144.42/1375893.38)
SimEnuLoAw-c9-l9-m10-i(0,0)	0.914	0.0000	(0.00/ 1.00– 0.68/ 42.00)	(0.00/ 1.00– 1.82/21.00)	(0.00/ 2.00– 8.22/ 7406.38)
SimEnuLoAw-c16-l16-m40-i(0,0)	0.775	0.0017	(0.00/ 6.00– 11.90/ 4872.00)	(0.00/ 3.00– 3.91/33.00)	(0.00/ 5.00– 1338.37/1038388.00)
SimEnuLoAw-c30-l16-m99-i(0,0)	0.373	0.0126	(0.00/21.00–110.69/16705.00)	(0.00/ 9.00–11.23/70.00)	(0.00/63.45– 6340.33/1591202.50)
SimEnuLoCl-c9-l9-m10-i(0,0)	0.902	0.0000	(0.00/ 1.00– 0.69/ 37.00)	(0.00/ 1.00– 2.29/28.00)	(0.00/ 2.50– 3.91/ 390.89)
SimEnuLoCl-c16-l16-m40-i(0,0)	0.756	0.0000	(0.00/ 6.00– 5.58/ 119.00)	(0.00/ 3.00– 4.49/33.00)	(0.00/ 4.50– 14.36/ 2825.11)
SimEnuLoCl-c30-l16-m99-i(0,0)	0.376	0.0013	(0.00/21.00– 35.93/11574.00)	(0.00/10.00–12.07/77.00)	(0.00/30.67– 947.77/ 621121.63)
SimEnuLoClf-c9-l9-m10-i(0,0)	0.902	0.0000	(0.00/ 1.00– 0.69/ 38.00)	(0.00/ 1.00– 2.29/28.00)	(0.00/ 2.50– 3.92/ 401.00)
SimEnuLoClf-c16-l16-m40-i(0,0)	0.756	0.0000	(0.00/ 6.00– 5.39/ 122.00)	(0.00/ 3.00– 4.49/33.00)	(0.00/ 4.50– 14.22/ 2543.00)
SimEnuLoClf-c30-l16-m99-i(0,0)	0.375	0.0013	(0.00/21.00– 37.62/ 9978.00)	(0.00/10.00–12.07/77.00)	(0.00/30.73– 930.32/ 604716.19)

Table 13: Additional statistics: for each algorithm are given statistics (minimum/median–average/maximum) about the average number of cells in the fringe of a game's CSPs, the maximum number of cells in the fringe of a game's CSPs, and the number of uncovered cells at the end of the game

	nbCellsInFringe	maxCellsInFringe	nbUncoveredCells	maxNbSolutions
SimRan-c9-l9-m10-i(0,0)	(0.00/ 0.00– 0.00/ 0.00)	(0.00/ 0.00– 0.00/ 0.00)	(1.00/ 71.00– 58.54/ 71.00)	(0.00/ 0.00– 0.00/ 0.00)
SimRan-c16-l16-m40-i(0,0)	(0.00/ 0.00– 0.00/ 0.00)	(0.00/ 0.00– 0.00/ 0.00)	(1.00/207.00–139.26/216.00)	(0.00/ 0.00– 0.00/ 0.00)
SimRan-c30-l16-m99-i(0,0)	(0.00/ 0.00– 0.00/ 0.00)	(0.00/ 0.00– 0.00/ 0.00)	(1.00/ 19.00– 86.33/381.00)	(0.00/ 0.00– 0.00/ 0.00)
SimEnuAwLo-c9-l9-m10-i(0,0)	(0.00/ 4.00– 4.77/ 23.93)	(0.00/ 4.00– 6.02/ 48.00)	(1.00/ 71.00– 66.47/ 71.00)	(0.00/ 3.00– 120.85/ 840000.00)
SimEnuAwLo-c16-l16-m40-i(0,0)	(0.00/ 8.67– 9.38/ 52.10)	(0.00/12.00–13.86/108.00)	(1.00/216.00–181.96/216.00)	(0.00/ 8.00– 18132.62/3428138.00)
SimEnuAwLo-c30-l16-m99-i(0,0)	(0.00/17.50–18.00/101.48)	(0.00/31.00–31.58/170.00)	(1.00/322.00–214.68/381.00)	(0.00/360.00–175373.06/3401008.00)
SimEnuLoAw-c9-l9-m10-i(0,0)	(0.00/ 4.00– 4.82/ 21.23)	(0.00/ 4.00– 6.02/ 38.00)	(1.00/ 71.00– 66.65/ 71.00)	(0.00/ 3.00– 29.20/ 56700.00)
SimEnuLoAw-c16-l16-m40-i(0,0)	(0.00/ 8.75– 9.25/ 35.64)	(0.00/12.00–13.27/ 53.00)	(1.00/216.00–185.93/216.00)	(0.00/ 8.00– 6067.23/3432667.00)
SimEnuLoAw-c30-l16-m99-i(0,0)	(0.00/17.14–16.73/ 58.48)	(0.00/30.00–27.92/ 81.00)	(1.00/378.00–245.05/381.00)	(0.00/228.00– 36097.33/3697149.00)
SimEnuLoCl-c9-l9-m10-i(0,0)	(0.00/ 4.50– 4.95/ 20.46)	(0.00/ 6.00– 6.22/ 31.00)	(1.00/ 71.00– 65.82/ 71.00)	(0.00/ 3.00– 7.26/ 1359.00)
SimEnuLoCl-c16-l16-m40-i(0,0)	(0.00/ 8.33– 8.90/ 35.64)	(0.00/12.00–12.52/ 47.00)	(1.00/216.00–182.41/216.00)	(0.00/ 6.00– 43.73/ 15312.00)
SimEnuLoCl-c30-l16-m99-i(0,0)	(0.00/15.41–15.30/ 58.48)	(0.00/28.00–25.87/ 81.00)	(1.00/378.00–246.37/381.00)	(0.00/108.00– 6900.93/3281169.00)
SimEnuLoClf-c9-l9-m10-i(0,0)	(0.00/ 4.50– 4.95/ 20.46)	(0.00/ 6.00– 6.23/ 31.00)	(1.00/ 71.00– 65.82/ 71.00)	(0.00/ 3.00– 7.23/ 1638.00)
SimEnuLoClf-c16-l16-m40-i(0,0)	(0.00/ 8.33– 8.90/ 35.64)	(0.00/12.00–12.52/ 47.00)	(1.00/216.00–182.41/216.00)	(0.00/ 6.00– 43.20/ 15312.00)
SimEnuLoClf-c30-l16-m99-i(0,0)	(0.00/15.42–15.31/ 58.48)	(0.00/28.00–25.87/ 81.00)	(1.00/378.00–246.36/381.00)	(0.00/108.00– 6692.55/3117115.00)

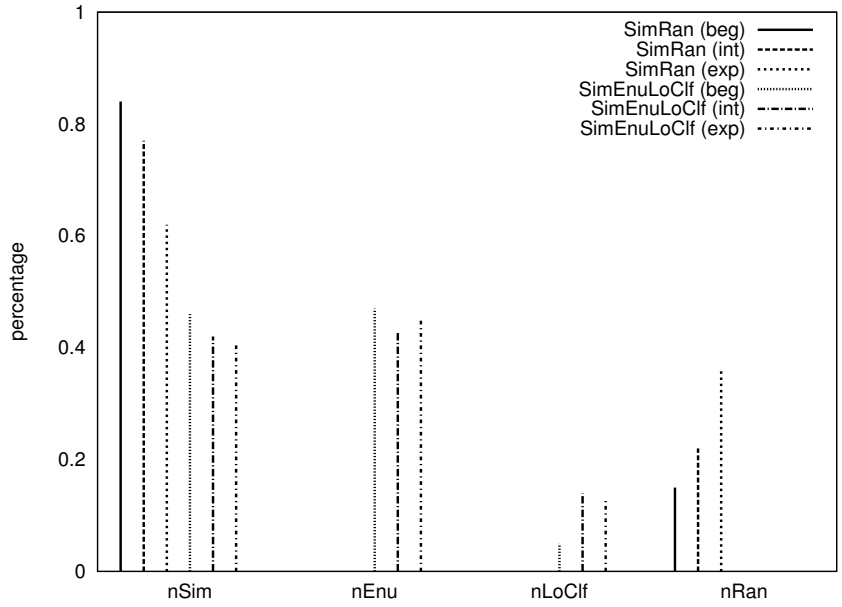


Figure 3: Usage distribution over the building block algorithms of 2 strategies in the 3 difficulty levels

As expected, the simple rule makes most of the decisions, followed by the enumeration algorithm (for `SimEnuLoClf`) and random choices (for `SimRan`). Only a minority of cases require `SimEnuLoClf` to make a guess through `LoClf`.

One can also observe on *alternate* levels (Figure 4) that proportions are almost identical for intermediate(-alt) and expert games, as for mine densities (20.5% and 20.6%). As could be expected, higher mine densities lead to more difficult games requiring more guessing.

5 Discussion

5.1 Computer vs. Human Minesweeper

An interesting point is that playing Minesweeper is not the same thing for humans and for computers. Human players are typically evaluated based on how fast they can solve games rather than on how often they solve games. While speed and accuracy have to be compromised —playing fast requires taking risks— this is thus usually done in different ways whether the player is a human or a computer. Reasons for these different choices may be that (i) estimating the average success rate requires too many games, and (ii) optimizing for success rate would lead to taking a lot more time for solving each game.

The objective of minimizing best resolution times also has the drawback that speed depends on the difficulty of a minefield. Some cases can be solved in very few clicks, thus essentially by chance. This led to introduce the requirement that a score is valid only if it has been achieved on a grid require a minimum number of clicks to be uncovered. This threshold depends on the game difficulty (grid size and number of mines). From a different viewpoint, it is a challenge in itself to solve grids with as few clicks as possible.

Additionally, while both types of players try to "think as well as possible", humans have the

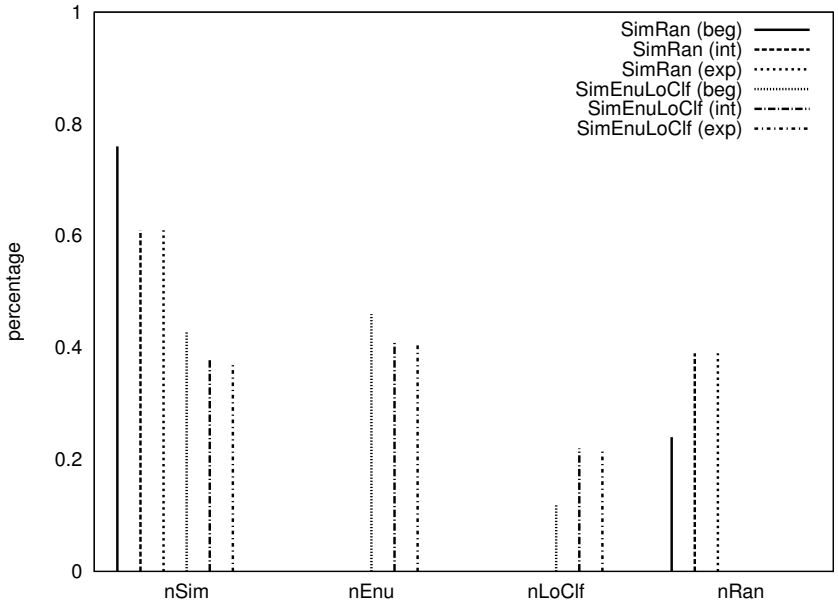


Figure 4: Usage distribution over the building block algorithms of 2 strategies in the 3 *alternate* difficulty levels

additional difficulty to also optimize their physical interactions, i.e., to control their mouse as fast and accurately as possible.

On the side of computer minesweeper, time is also an issue that constrains to use approximate solvers. At first sight, our experiments seem to show that good decision rules —rules to decide which unsafe cell to probe— typically lead to easier inference problems to solve. This is misleading since some strategies are limited by the maximum solution time for the enumeration algorithm. It is expected that, if time were not a constraint, then the best strategies would in fact involve harder inference problems.

5.2 Evaluation

An ideal situation would be to optimize for time only, being sure that contestants are success-rate optimal. Yet, one cannot guarantee that a human player is success-rate optimal —because humans make mistake— , and there are for now no provably success-rate optimal algorithms. One cannot optimize for success rate only without caring about time: this could lead to very long competitions if winning means making more accurate computations than competitors.

So, the problem is how to combine both criteria: success rate and time, possibly turning one of them into a constraint. A first suggestion is to always put a time constraint: as suggested above not setting a time limit could lead to some games taking too much time. One can then optimize, for example:

- success rate alone,
- time alone, under a success rate constraint,
- success rate first, time second (to break ties).

5.3 Going Further

There are various directions of research regarding heuristic-based strategies:

- improve the enumeration algorithm (what could be easy for the version used in our experiments, e.g., splitting the problem in independent problems whenever possible);
- study the percolation problem in more depth; and
- search for information-based heuristics —i.e., attempting to clarify the status of hidden cells—, knowing that being greedy with respect to the information gained is often an efficient strategy.

Another important direction would be to really try optimizing decisions over multiple time steps, i.e., considering the problem as a sequential decision-making one. Monte-Carlo Tree Search techniques could be good candidates for that, and could benefit from informative and fast heuristic approaches. But then, good heuristics may have a lower probability of success, but run very quickly.

References

M. Araya-López, V. Thomas, O. Buffet, and F. Charpillet. A closer look at MOMDPs. In *Proceedings of the Twenty-Second IEEE International Conference on Tools with Artificial Intelligence (ICTAI-10)*, 2010.

K. Bayer, J. Snyder, and B. Y. Choueiry. An interactive constraint-based approach to Minesweeper. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'06)*, pages 1933–1934, 2006.

R. Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

B. Bonet and H. Geffner. Solving POMDPs: RTDP-Bel vs. point-based algorithms. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009.

A. R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Department of Computer Science, Providence, RI, 1998.

L. P. Castillo and S. Wrobel. Learning Minesweeper with multirelational learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.

R. Collet. Playing the Minesweeper with constraints. In *Multiparadigm Programming in Mozart/OZ: Second International Conference, MOZ*, volume 3389 of *LNCS*, pages 251–262. Springer, 2004.

Y.-C. Huang, B. Selman, and H. Kautz. Learning declarative control rules for constraint-based planning. In *Proceedings of the International Conference on Machine Learning (ICML'00)*, 2000.

R. Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.

D. Knuth. The computer as a master mind. *Journal of Recreational Mathematics*, 9(1):1–6, 1976–77.

B. Kooi. Yet another mastermind strategy. *ICGA Journal*, 28(1):13–20, 2005.

T. Kostka. On the first click in microsoft's minesweeper, 2006. URL `http://devtk.com/minesweeper/firstclick.html`.

H. Kurniawati, D. Hsu, and W. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems IV*, 2008.

Jon McLoone. 25 best hangman words, August 2010. URL `http://blog.wolfram.com/2010/08/13/25-best-hangman-words/`.

MineSweeper. Authoritative Minesweeper: Articles, papers, reviews, reports, 2011. URL `http://www.minesweeper.info/articles/index.html?category=Math`.

G. Monahan. A survey of partially observable Markov decision processes. *Management Science*, 28:1–16, 1982.

E. Mossel. The minesweeper game: Percolation and complexity. *Journal of Combinatorics, Probability and Computing*, 11:487–499, September 2002. doi: 10.1017/S096354830200528X.

P. Nakov and Z. Wei. Minesweeper, #minesweeper. Technical report, EECS, Berkeley, 2003.

S.C.W. Ong, S.W. Png, D. Hsu, and W.S. Lee. POMDPs for robotic tasks with mixed observability. In *Proceedings of Robotics: Science and Systems V (RSS'09)*, 2009.

K. Pedersen. The complexity of Minesweeper and strategies for game playing. Technical report, Department of Computer Science, University of Warwick, 2004.

J. Pineau, G. Gordon, and S. Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 27:335–380, 2006.

S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 32:663–704, 2008.

M. Sebag and O. Teytaud. Combining myopic optimization and tree search: Application to MineSweeper. In *Proceedings of the Sixth Learning and Intelligence Optimization Conference (LION)*, 2012. URL `http://hal.inria.fr/hal-00712417`.

H.S. Sim, K.-E. Kim, J.H. Kim, D.-S. Chang, and M.-W. Koo. Symbolic heuristic search value iteration for factored POMDPs. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence (AAAI'08)*, 2008.

T. Smith and R.G. Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

M. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005. URL `http://www.aaai.org/Papers/JAIR/Vol24/JAIR-2406.pdf`.

Ian Stewart. Million-dollar Minesweeper. *Scientific American*, page 80, October 2000.

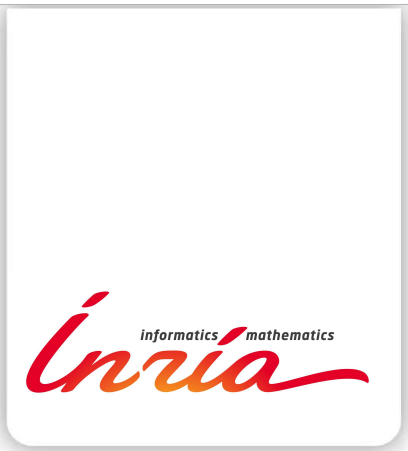
M. Vomlelova and J. Vomlel. Applying Bayesian networks in the game of Minesweeper. In *Proceedings of the Twelfth Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty*, pages 153–162, September 2009.

Eric W. Weisstein. Mastermind, 2011. URL `http://mathworld.wolfram.com/Mastermind.html`. From MathWorld–A Wolfram Web Resource.

wikipedia. Hangman (game), 2011a. URL `http://en.wikipedia.org/wiki/Hangman_(game)`.

wikipedia. Mastermind (board game), 2011b. URL `http://en.wikipedia.org/wiki/Mastermind_(board_game)`.

wikipedia. Minesweeper (video game), 2011c. URL `http://en.wikipedia.org/wiki/Minesweeper_(video_game)`.



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399